Programming 2
Tutorial and Practical 7

Tutorial Session

This tutorial will focus on the concepts and algorithms required to implement a basic event-driven graphical application. It is not necessary to work out the source code at this stage; the actual implementation can be left for the laboratory session.

1. Consider writing a small, simple Java Swing program (utilising a `JFrame` for the application window) that draws a few basic shapes (circles/ovals, squares/rectangles, and lines) to the screen. Although this is a very basic exercise, it will introduce two challenges; drawing in a `JFrame`, and setting up the application.

   Drawing

   a. How is freeform drawing performed on a component?
   b. Why must a `JPanel` be used as a drawing canvas, rather than the `JFrame` itself? What is the difference in how those two classes are implemented, and what implications does this have on how they can be used?
   c. Since we're using this `JPanel` just to draw shapes with very specific parameters, consider extending `JPanel` to create a "shape drawing" component.
      i. Is there any specific information that must be stored in the class?
      ii. What methods should be provided to the user of the class?
      iii. What information must be passed in to draw each shape type?

   Setting Up the Application

   d. How is a `JFrame` added to a Java application? How can the code be arranged?
   e. What happens when the `JFrame` is closed?
   f. How can the application be made to quit?

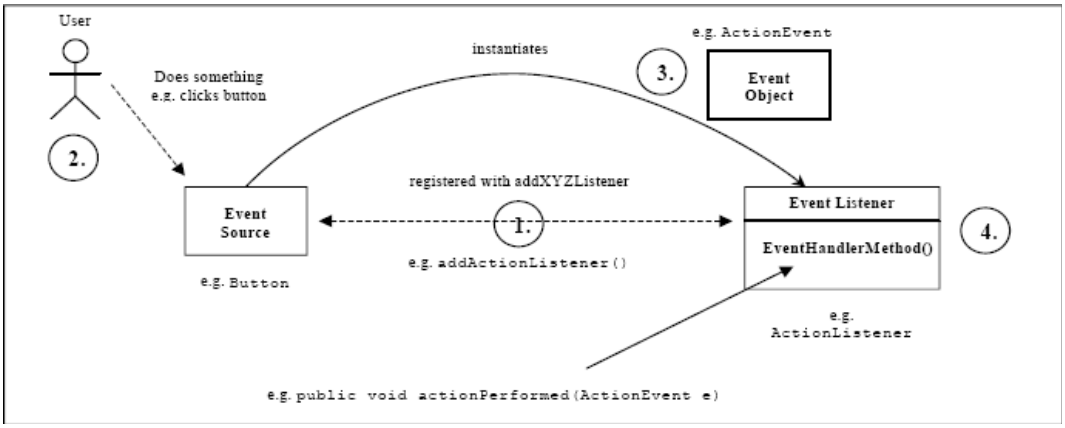
2. By now we have created a `JFrame`, and used a `JPanel` for our drawing – a panel that has been extended to make drawing specific shapes (with certain attributes) easy. The logical step from here is to add some interactivity, so that the user can choose what gets drawn, and where.

   Consider adding some buttons to the `JFrame` (*Square*, *Circle*, *Line*, etc.), for the user to select which kind of shape they would like to draw. When the user clicks a certain place in the drawing area, the selected shape will be drawn.

   Interactivity with Events

   a. How can the buttons be made to work?
      i. Which event type is most appropriate? Which listener? (and method within the listener, where appropriate)
      ii. What is the event source? Where should the listener go?
      iii. What will be the implementation of the listeners?
   b. Consider aspects (i) to (iii) for handling the situation where the mouse is clicked in the drawing area.

   For each situation requiring event handling consider the requirements in terms of the event-handling model shown in the following diagram (this model was covered in the lecture):



Practical Session

In the practical session, implement the shape drawing program discussed in the tutorial.

When writing the simple `JFrame`-based implementation as per tutorial question 1, first draw your attention towards extending the `JPanel` class into a specialised shape drawing component, and then use this in your `JFrame`-based application.

Even if you're confident that you could start working on the interactive `JFrame`-based implementation, consider writing this simpler version anyway. It will provide a useful apparatus for testing your drawing component in isolation. (Trying to test this component later may prove troublesome: if errors occur, were they due to the drawing component, or how your application interacts with it?)

Finally, move onto the `JFrame` implementation with buttons and mouse interaction as discussed in tutorial question 2. (If your earlier implementation was implemented cleanly, it should be a straightforward matter to add buttons and event management code.)

If you have time, experiment with implementing the event listeners in different places:

- Internal event listeners
- External event listeners
- Inner classes

More information on how to organise event listeners is provided in the "Advanced Event Management" sub-topic of the online learning materials of Blackboard.